Near-Miss Model Clone Detection for Simulink Models

Manar H. Alalfi James R. Cordy Thomas R. Dean Matthew Stephan Andrew Stevenson

School of Computing, Queen's University, Kingston, Canada
{*alalfi, cordy, dean, stephan, andrews*}*@cs.queensu.ca*

***Abstract*—This paper describes our plan to adapt mature code-based clone detection techniques to the efficient identification of near-miss clones in models. Our goal is to leverage successful source text-based clone detection techniques by transforming graph-based models to normalized text form in order to capture semantically meaningful near-miss results that can help in further model analysis tasks. In this position paper we present a first example, adapting the NiCad code clone detector to identifying near-miss Simulink model clones at the "system" granularity. In current work we are extending this technique to the Simulink (entire) "model" and (more refined) "block" granularities as well.**

I. Introduction

Model clone detection refers to the process of identifying similar or identical fragments in higher-level software models based on some measure of similarity. While its counterpart, code clone detection, is a mature and established area of research [1], model clone detection is relatively new and has not been investigated as thoroughly. This is an issue for two reasons: first, model driven development is rapidly becoming a dominant method of new software development, and second, the potential impact of identifying redundancy at higher levels is greater than at lower levels.

Not surprisingly, approaches to model clone detection to this point have primarily utilized graph-based techniques [2, 3, 4]. That is, they represent the models as nodes and edges and use variations of subgraph matching techniques to find clones. While natural and efficient for exact matching in visual models, these methods have had less success in near-miss clone detection [4]. In this paper, we propose a method for leveraging existing near-miss textual code analysis techniques, such as the *NiCad* code clone detector and the *LDA* topic model [5], in order to detect near-miss model clones based on the hybrid syntactic approach of NiCad.

The anticipated contributions of our approach are:

- Efficient detection of not only type 1 (exact) and type 2 (renamed) clones, but also type 3 (near-miss) model clones. Existing approaches handle types 1 and 2, but have difficulty with near-miss.
- We plan for at least three different levels of syntactic granularity: Simulink (entire) "model", (sub-) "system" and (detailed) "block". Existing approaches concentrate on the block level of granularity.
- Our approach returns (near-miss) syntactic and semantic clones. Existing approaches use subgraph or semantic matching [6] rather than syntactic structure.

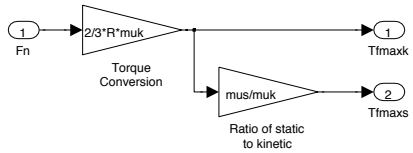

Figure 1. A type 1 (exact) model clone - the *Friction Mode* subsystem, which occurs in both the *Sldemo_Clutch* and the *Sldemo_Clutch_if* example models. NiCad similarity 100%

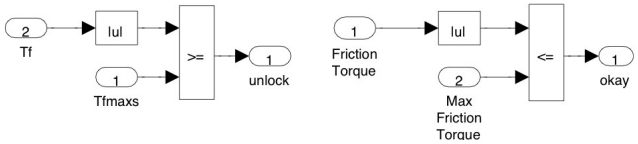

Figure 2. A type 2 (renamed) model clone - the *Required Friction for Lockup* subsystem (Left) and the *Break Apart Detection* subsystem (Right), both in the *Sldemo_Clutch_if* example model. NiCad similarity 85%

In this paper we demonstrate our early results with examples of type 1, 2 and 3 clone identification in Simulink example models at the subsystem level of syntactic granularity.

II. Approach

We extended *NiCad* [7], a clone detection tool based on parsing, normalizing, and text-comparing syntactic fragments, to model clone detection. *NiCad* is explicitly designed to allow for unexpected differences in near-miss clones up to a given difference threshold. It is based on a plugin architecture that allows for new languages and normalizations, which we used to extend it to Simulink model clone detection.

A. Clone Types

Code clone detection techniques can be categorized according to the types of clones they can identify [1]. Because our approach is an adaptation of one of the code cloning techniques, we adopt the same categorization for model clones. In our first experiment we have identified three types of model clones at the Simulink subsystem level:

1) Type 1 (Exact) Model Clones: Identical model fragments except for variations in layout and formatting. Figure 1 shows an example in which two different models, *Sldemo_Clutch* and *Sldemo_Clutch_if*, include the identical subsystem *Friction Mode*.

2) Type 2 (Renamed) Model Clones: Structurally identical model fragments except for variations in labels, values, types, layout and formatting. Figure 2 shows an example type 2 clone of two different subsystems (*Required Friction for Lockup* and *Break Apart Detection*) in the *Sldemo_Clutch* example model.

978-1-4673-1795-5/12/$31.00 © 2012 IEEE

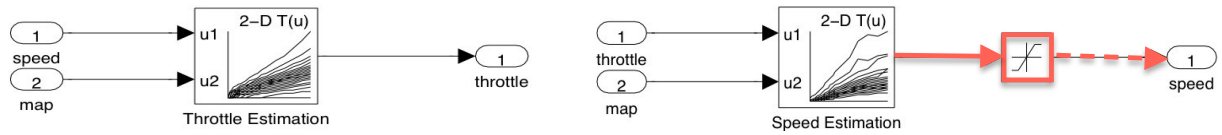

Figure 3. A type 3 (near-miss) model clone - the *Throttle.throttle_estimate* subsystem (Left) and *Speed.speed_estimate* subsystem (Right) of the *sldemo_fuelsys* model. NiCad similarity 66%

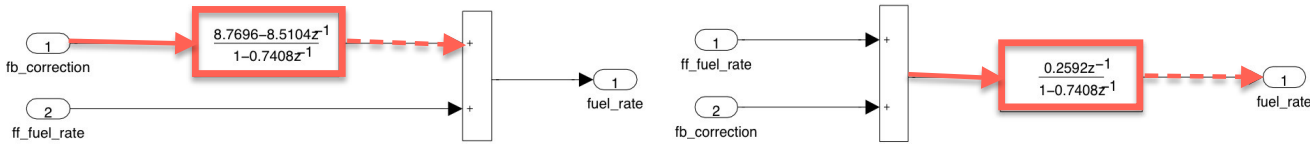

Figure 4. A type 3 (near-miss) model clone - the *Low mode* subsystem (Left) and *Rich mode* subsystem (Right) of the *sldemo_fuelsys* example model. NiCad similarity 82%

3) Type 3 (Near-Miss) Model Clones: Model fragments with further modifications, such as changes in location with respect to other model fragments and small additions or removals of blocks or lines in addition to variations in labels, values, types, layout and formatting. Figure 3 shows a type 3 clone between the *Throttle.throttle_estimate* and the *Speed.speed_ estimate* subsystems of the *sldemo_fuelsys* example model. A new block and line have been added as well as naming and attribute changes to other blocks and lines. Figure 4 shows a second type 3 example in which the structure of the subsystem has been modified to move a block to another location in relation to other blocks.

B. Clone Granularities

We have identified three levels of granularity for Simulink models:

1) Model Granularity: Entire Simulink models as clones. Simulink models consist of (sub-) systems, which themselves are built up from blocks and lines. At this level of granularity we can evaluate similarity of whole models such as *sldemo_clutch* and *sldemo_clutch_if*. We are currently developing a NiCad plugin for this granularity.

2) System Granularity: On the Simulink "system" (subsystem) level, we have identified clones in two dimensions in the example models:

- Exact subsystem clones across two different models, for example the *Friction Model* subsystem in both the *sldemo_clutch sldemo_clutch_if* models (Figure 1).
- Near-miss subsystem clones within a single model, for example the *Required Friction for Lockup* and *Break Apart Detection* subsystems of the *Sldemo_Clutch_if* example model (Figure 2).

3) Block Granularity: Blocks are the most fine-grained elements of Simulink models. However, blocks can also contain subsystems, which represent a group of blocks and lines that work together to provide a specific functionality. A NiCad plugin for this granularity is under development.

III. Related Work

Deissenboeck et al. [2], Pham et al. [4], and Peterson [3] all employ graph-based techniques for model clone detection. We discuss how they can be contrasted with our approach in [8]. Al-Batran et al. [6] identify a number of semantics-preserving transformations that allow for detection of semantically equivalent clones. We may be able to incorporate their work into our approach.

IV. Conclusion and Future Work

We have presented some initial results on adapting a text-based code clone detection technique to identify model clones at the Simulink "system" granularity. We are currently adapting the technique to identify clones at the model and block levels as well. We plan to enhance our method with LDA topic models in order to add semantic comparison. We also plan to run an experiment to compare existing model clone detectors to our new method, as outlined in [8].

Acknowledgements

This work is supported in part by NSERC, as part of the NECSIS Automotive Partnership with General Motors, IBM Canada and Malina Software Corp.

References

[1] C. Roy, J. Cordy, and R. Koschke, "Comparison and evaluation of code clone detection techniques and tools: A qualitative approach," *Science of Computer Programming*, vol. 74, no. 7, pp. 470–495, 2009.

[2] F. Deissenboeck, B. Hummel, E. Jurgens, B. Schatz, S. Wagner, J. Girard, and S. Teuchert, "Clone detection in automotive model-based development," in *ICSE*, 2009, pp. 603–612.

[3] H. Petersen, "Clone detection in Matlab Simulink models," Master's thesis, Tech. Univ. of Denmark, 2012, iMM-M.Sc.-2012-02.

[4] N. Pham, H. Nguyen, T. Nguyen, J. Al-Kofahi, and T. Nguyen, "Complete and accurate clone detection in graph-based models," in *ICSE*, 2009, pp. 276–286.

[5] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *J. Mach. Learn. Res.*, vol. 3, pp. 993–1022, 2003.

[6] B. Al-Batran, B. Schätz, and B. Hummel, "Semantic clone detection for model-based development of embedded systems," *Model Driven Engineering Languages and Systems*, pp. 258–272, 2011.

[7] C. Roy and J. Cordy, "NICAD: Accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization," in *ICPC*, 2008, pp. 172–181.

[8] M. Stephan, M. Alafi, A. Stevenson, and J. Cordy, "Comparison of model clone detection approaches," in *IWSC*, 2012, (to appear).